BINDURA UNIVERSITY OF SCIENCE EDUCATION

FACULTY OF SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE

Implementing Scheduled Synthetic Monitoring for Near Real-Time Cloud Service Availability Tracking: *A Case Study of Central African Building Society*

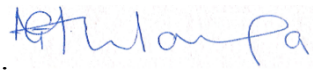
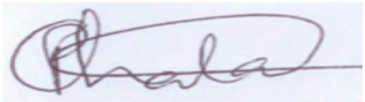
BY

BAKO ALEXANDER (B202176B)

This dissertation is submitted in partial fulfilment of the requirements of the Bachelor of Science honours degree in Software Engineering

APPROVAL FORM

The undersigned certify that they have supervised the student Bako Alexander's dissertation entitled " Implementing Scheduled Synthetic Monitoring for Near Real-Time Cloud Service Availability Tracking: *A Case Study of Central African Building Society*" submitted in Partial fulfilment of the requirements for the Bachelor of Software Engineering Honours Degree of Bindura University of Science Education.

A. BAKO…….		20…/…06/ 2025…..
STUDENT	SIGNATURE	DATE
…G. MHLANGA……....	 …….	…06…/…08./2025…
SUPERVISOR	SIGNATURE	DATE
…P CHAKA…….	 .	06…/…08/.2025……
CHAIRPERSON DATE	SIGNATURE	DATE
……..	……..	……/……/……..
EXTERNAL EXAMINER	SIGNATURE	DATE

ACKNOWLEDGEMENT

I would like to begin by expressing my deepest gratitude to God for the opportunity to work on this research. I would like to express my heartfelt gratitude to all those who contributed to the success of this research. Special thanks go to the management and staff of OLD MUTUAL-CABS bank Zimbabwe for graciously allowing me to conduct this study within their institution and for providing valuable insights and support throughout the research period. I am equally thankful to Bindura University of Science Education for the academic guidance, resources, and mentorship that made this work possible. I also appreciate the collaboration and encouragement from colleagues and friends who offered feedback and moral support during the course with the project. Your collective input has been instrumental in shaping this study into what it is today. I would like to acknowledge my lecturers in the Department of Computer Science and Engineering for their patience, support, and for equipping me with the knowledge and skill sets necessary for this project to succeed. I am particularly thankful to my lecturers, Mr. Hove, Mr. Mhlanga, and Mr. Chaka, for their unwavering guidance, encouragement, and insightful feedback, which greatly enriched the quality of this work. Finally, I want to express my gratitude to my parents for their unconditional love and unwavering support throughout my academic journey. They have been my financial and emotional pillars and have encouraged me to be the best version of myself.

ABSTRACT

In an era where cloud infrastructure underpins critical banking operations, ensuring continuous service availability is paramount, particularly for financial institutions operating in dynamic and high-risk environments. This study examines the implementation of scheduled synthetic monitoring as a contemporary, proactive approach to tracking cloud service availability, utilizing the Central African Building Society (CABS) as a case study. CABS currently relies on manual monitoring techniques such as SSH logins, ping, and grep commands, which are reactive, time-consuming, and prone to human error. This project introduces an automated, scheduled synthetic monitoring framework designed to simulate user interactions, detect availability issues in near real-time, and generate actionable alerts before customers are affected. Through comparative analysis, we evaluate the performance, reliability, and operational efficiency of synthetic monitoring against traditional methods. The study also explores the strategic implications of adopting intelligent monitoring practices in the financial sector, including regulatory compliance, risk mitigation, and service-level assurance. By transitioning from manual to automated monitoring, the proposed solution aims to enhance service resilience, reduce downtime, and support digital transformation efforts across financial institutions in emerging economies.

Table of Contents

TABLE OF FIGURES

Definition of terms

- URL – Uniform Resource Locator
- HTTP - Hypertext Transfer Protocol
- HTTPS - Hypertext Transfer Protocol Secure
- SaaS – Software as a Service
- PaaS - Platform as a Service
- IaaS - Infrastructure as a Service
- SLAs - Service Level Agreements
- Web Service - a software system designed to enable communication and data exchange between applications or devices over a network, typically the Internet
- Endpoint - is a specific URL or address through which a client can access a service or resource hosted on a server. It acts as the interface between the client (e.g., applications, users, or devices) and the server, enabling communication and data exchang

Chapter 1: Problem Identification

Introduction

In today's digitally interconnected world, cloud services are at the core of many applications and business operations. These services facilitate seamless communication and data exchange between different systems, enabling functionalities such as online transactions, cloud storage, and API-based integrations. As the reliance on cloud services continues to grow, ensuring their reliability becomes increasingly critical. Effective monitoring is essential for ensuring user satisfaction, maintaining SLAs (Service Level Agreements), and enabling proactive problem resolution. Over the years, this field of monitoring hasn't been pursued by many organisations due to its complexity, and others ignore it since it is not a productive side of the business, among other reasons. However, in today's business, everything is now customer-centric, which means service delivery is now the spotlight. In order to gain our customers' trust and confidence, organisations are supposed to offer efficient and reliable services to their clients. This includes having maximum service uptime possible and having quicker resolutions in case of faults and incidents. This brings us to the issue of monitoring to ensure that service availability is guaranteed.

This project focuses on a key aspect of maintaining robust cloud service monitoring of their endpoints. A "cloud service endpoint" is a specific network address (usually a URL) that acts as the entry point for accessing a particular cloud service, essentially providing a way to connect to and utilize the features of that service within a cloud platform (AWS, 2025). Failures or inconsistencies at these endpoints can lead to downtime, degraded performance, and disruptions in service delivery, potentially causing significant financial and reputational losses. The goal of this project is to develop a systematic approach to monitor the cloud services of CABS and ensure that these services are delivered to the clients as expected at all times. By leveraging advanced monitoring techniques and reliability metrics, this project aims to provide actionable insights into the performance and availability of cloud services. This will help organizations proactively address issues, enhance system resilience, and deliver consistent user experiences. Through this initiative, we aim to bridge the gap between theoretical reliability assessments and practical monitoring solutions, offering a

comprehensive framework that supports both developers and IT administrators in maintaining high-quality cloud services.

1.1 Background of the study

In today's fast-paced digital economy, cloud computing has become the backbone of modern banking services. Financial institutions rely heavily on cloud-based solutions to offer secure, scalable, and efficient services to their customers. Whether it is online transactions, mobile banking, or digital wallets, the availability of cloud services plays a crucial role in ensuring a seamless banking experience (Marston et al., 2011). However, cloud outages, latency issues, and unexpected downtimes can severely impact banking operations, leading to financial losses, reputational damage, and customer dissatisfaction (Armbrust et al., 2010). Real-world incidents show how devastating these failures can be. Here in Zimbabwe, it had been the same for financial institutions, losing large amounts of money due to service failures and downtimes. Specifically, CABS lost an estimated figure of one hundred thousand dollars according to analysis by its marketing department (CABS, 2024). In December 2024, Intesa Sanpaolo, Italy's largest bank, suffered a massive online banking outage due to system failures triggered by intense traffic (Reuters, 2024). Thousands of customers were locked out of their accounts, unable to complete transactions or access financial services. This is not an isolated case; cloud failures happen more often than people realize, affecting businesses that rely on them for daily operations (Cetin et al., 2021). The problem is even more pronounced in multi-tenant cloud environments, where service availability, security, and compliance become complex issues, particularly in regions with less-developed financial infrastructure (Yeboah-Boateng et al., 2016).

The increasing frequency of cloud outages has raised concerns among financial institutions, regulators, and consumers. Service disruptions not only cause financial losses but also erode customer confidence in digital banking systems. To mitigate these risks, banks are adopting multi-cloud strategies, enhancing redundancy measures, and developing robust disaster recovery plans and proactive monitoring solutions that can detect faults before they impact customers (McKinsey & Company, 2022). Furthermore, regulatory frameworks are evolving to ensure cloud resilience in banking. The European Banking Authority (EBA) has introduced stricter guidelines on cloud outsourcing risk management, emphasizing transparency, accountability, and contingency planning (EBA, 2022). Similarly, the U.S. Federal Reserve

has recommended stress-testing cloud infrastructures to assess their resilience against potential failures (Federal Reserve, 2023).

Despite the availability of monitoring tools, research shows that 18% of organisations have no monitoring tools, and others use manual checks such as SSH, grep, and ping (Damian et al., 2020). Such approaches cannot pick up faults promptly. As a result, incidents are discovered by clients rather than application owners. This reactive nature exposes organizations to risks of service interruptions, which can result in financial losses, reputational damage, and a diminished user experience. In addition, the complexity of modern service architectures, coupled with the increasing adoption of microservices and API-driven systems, has created a need for advanced, proactive monitoring solutions.

Globally, cloud service outages have been on the rise, with increasing reliance on cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud. A report by Gartner (2022) indicated that global cloud downtime incidents have surged by 35% over the past five years, with financial institutions among the most affected sectors. This growth is attributed to various factors, including cyberattacks, misconfigurations, software bugs, and infrastructure failures. Regulatory bodies, such as the Bank for International Settlements (BIS), have raised concerns about the systemic risks posed by cloud concentration in banking (BIS, 2021). The dependency of multiple financial institutions on a few major cloud providers creates a single point of failure, making the industry vulnerable to large-scale disruptions. As a result, central banks and financial regulators are increasingly scrutinizing cloud resilience strategies and advocating for multi-cloud deployments to mitigate risks (European Central Bank, 2022).

Given the critical nature of banking services, real-time or near real-time monitoring of cloud service availability is essential. Traditional monitoring methods often rely on periodic checks, which may not be fast enough to detect service disruptions before they affect customers. Banks need a more proactive approach that provides real-time insights into cloud performance, allowing for quick decision-making and mitigation of risks (Buyya et al., 2010). Implementing near real-time cloud monitoring ensures that banks can quickly respond to issues, minimize downtime, and enhance the overall resilience of their services (Rimal et al., 2009).

Cloud service providers, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, offer built-in monitoring tools. However, these tools may not always align with the specific needs of banking institutions, particularly in terms of compliance, security, and integration with existing banking systems (Hashem et al., 2015). Therefore, banks require customized monitoring solutions that provide continuous visibility into cloud availability and performance metrics while ensuring regulatory compliance and data protection.

This study focuses on the development of a near-real-time cloud services availability monitoring system tailored for banking services. By leveraging monitoring techniques such as the RESTful GET method, POST method, and Socket connection trials to test the availability of test complete servers and services. Monitoring is done from the client's end, allowing the bank to note the quality of service being delivered to its customers. This is cemented by keeping track of matrices such as response time and failed job count. Banks can gain deeper insights into service health and potential failures before they escalate (Garg, Versteeg & Buyya, 2011). The goal is to enhance service reliability, improve customer trust, and ensure uninterrupted banking operations in an increasingly cloud-dependent environment.

Cloud service outages in the banking sector pose significant challenges, from disrupting financial transactions to undermining regulatory compliance and customer trust. As the banking industry continues to embrace cloud technology, it is imperative to address these vulnerabilities through strategic resilience planning, regulatory oversight, and technological innovation. By adopting proactive monitoring measures, banks can ensure uninterrupted service delivery, safeguarding financial stability in an increasingly digitalized world.

1.2 Statement of the problem

CABS had been facing challenges of cloud service disruptions and service failures due to traditional monitoring techniques. The bank usually relies on manual checks such as SSH, grep, and ping, but these methods were falling short as they failed to notice service outages on time. In 2024, the marketing department expressed its concern over losing customers to other banks due to frustrations from these devastating service outages. These monitoring techniques also created issues with false positive alerts, where the service appeared to be online from the bank's perspective, but the clients were unable to access it. As a result, problem resolution would be delayed, as the bank relied on customer reports to identify

service failures. Therefore, it became desirable to develop a more effective monitoring solution to address these challenges.

1.3 Research objectives

1. To create an app that:
    a. checks the availability of cloud services at 2-minute intervals
    b. Send SMS alerts if a cloud service is down
    c. Save downtimes and up times of cloud services in the MySQL database
    d. Visualize cloud services status on a dashboard
2. Determine if endpoint monitoring can be reliably utilized for monitoring of cloud services using statistical evaluation
3. Use statistical evaluation to measure the performance of the WS Monitor based on recorded responses and the ability to detect service outages in 8 week period .

1.4 Research questions

a. How can an app be designed to check the availability of cloud services at 2-minute intervals efficiently?
b. How can automated alerts be implemented to notify users immediately when a cloud service goes down?
c. What is the optimal way to store downtime and uptime data in a MySQL database for efficient retrieval and analysis?
d. How can cloud service status be visualized in an intuitive and user-friendly dashboard?
2. How accurate and reliable is endpoint monitoring in detecting cloud service availability issues?
3. How effective is the WS Monitor in detecting service outages over 30 days based on recorded responses and statistical performance evaluation?

1.5 Research propositions/hypothesis

H_1 endpoint is reliable for monitoring of cloud services

H_2 endpoint is not reliable for monitoring cloud services

1.6 Significance of the study

The significance of this study lies in addressing the growing demand for proactive monitoring solutions that go beyond traditional reactive methods. While existing tools can identify and address issues after the manual checks, they often fail to provide timely and comprehensive insights into long-term reliability. This gap leaves organizations vulnerable to service interruptions, data inconsistencies, and security risks. By developing a monitoring cloud service monitoring tool, this study contributes to minimizing downtime, improving fault tolerance, and enhancing overall service quality. A major focus of this research is simplicity and an easy procedure for monitoring cloud services to encourage even small businesses to monitor their services. This offers the potential to revolutionize how organizations monitor and manage their cloud services, fostering innovation in reliability engineering.

This research is particularly justified in the context of increasing reliance on digital systems, where even minor service disruptions can lead to significant financial and reputational losses. The findings of this study are expected to benefit software developers, IT administrators, and organizations by providing actionable strategies and tools to ensure consistent and reliable service delivery.

Significance to Customers

For banking customers, service availability is critical in ensuring seamless access to financial services such as fund transfers, bill payments, and loan applications. A cloud service monitoring tool enhances transparency by providing real-time updates on service status, minimizing frustration and uncertainty. It also helps in early detection of potential issues, ensuring uninterrupted banking experiences, and reinforcing customer trust in digital banking platforms.

Significance to the Bank

For financial institutions, downtime equates to revenue loss, reputational damage, and potential regulatory penalties. A cloud service monitoring tool enables proactive issue resolution by detecting anomalies before they escalate into full-blown outages. This improves operational efficiency, enhances service reliability, and ensures compliance with financial regulators' guidelines on risk management. Additionally, the insights gained from continuous monitoring can help banks optimize cloud infrastructure, reducing costs and improving performance.

Significance to the Universal Body of Knowledge

The development of a cloud service monitoring tool contributes to the broader field of knowledge by advancing research in cloud resilience, artificial intelligence-driven anomaly detection, and financial technology innovation. It provides empirical data on service availability trends, which can be used by researchers, policymakers, and technology developers to enhance cloud computing frameworks. Additionally, it facilitates collaboration between technology providers, academic institutions, and regulatory bodies to establish best practices in cloud reliability (Williams et al., 2023).

Significance to the Economy

At a macroeconomic level, ensuring cloud service availability in banking is vital for financial stability. The banking sector is a cornerstone of economic activity, facilitating transactions, investments, and economic growth. Persistent service disruptions can lead to market instability and a decline in investor confidence. A robust monitoring system helps prevent large-scale disruptions, supporting economic continuity and resilience in the face of technological challenges.

Cloud service outages in the banking sector pose significant challenges, from disrupting financial transactions to undermining regulatory compliance and customer trust. As the banking industry continues to embrace cloud technology, it is imperative to address these vulnerabilities through strategic resilience planning, regulatory oversight, and technological innovation. By adopting proactive measures, banks can ensure uninterrupted service delivery, safeguarding financial stability in an increasingly digitalized world.

1.7 Assumptions

- The data collected during the monitoring process is assumed to be accurate, complete, and representative of the typical behaviour of web service endpoints.
- It is assumed that the reliability of web service endpoints is critical to the overall functionality and performance of the cloud services they support.
- The study assumes that monitoring an endpoint allows us to detect which service is up or down.
- Key metrics such as response time, failed jobs, and availability are assumed to be accurate and sufficient indicators for evaluating the availability and reliability of the web service

- The study assumes that variations in user demand and traffic loads significantly impact the performance and reliability of web services
- The findings and recommendations derived from the study are assumed to be applicable across various industries and use cases that rely on cloud services.

1.8 Limitations/challenges

- Monitoring the reliability of endpoints in large-scale systems with numerous services and distributed architectures can be resource-intensive and may introduce latency.
- Modern web service environments are highly dynamic, with frequent updates, deployment changes, and traffic fluctuations, making it difficult to establish consistent monitoring baselines.
- Monitoring tools may need access to sensitive system or user data, raising concerns about data privacy and security, particularly in regulated industries.
- Collecting high-quality, comprehensive, and near-real-time data for monitoring web service endpoints can be challenging. Incomplete or inaccurate data may affect the overall findings of the study
- Implementing near-real-time monitoring and decision-making systems for availability can be difficult due to the need for immediate processing and action on vast amounts of data, which may require significant computational resources.

1.9 Scope/delimitation of the research

This research covers all three aspects of cloud services, that is, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service(IaaS), focusing on using scheduled synthetic monitoring to track the availability of services. To cover these three aspects, we will be using Test Complete servers, APIs, and software applications on the web used by the bank to deliver its services to its clients. In this research, we are not concerned about what could have caused faults, but only to determine whether a service is up or down. If it is not available, the aim is to notify the responsible people that the service has gone down. We should also have a record of these downtimes to report if it is ever needed.

Chapter 2: Literature Review

Introduction

The rapid advancement of cloud computing has led to its widespread adoption across industries, with financial institutions being among the most reliant on cloud infrastructure. The increasing reliance on cloud computing by financial institutions has necessitated the need for highly reliable and continuously available cloud services. Here in Zimbabwe, banks and other small financial institutions offer their services through the cloud for ease of doing business. The financial sector, characterized by high transaction volumes and strict regulatory requirements, demands near real-time monitoring of cloud service availability to mitigate risks associated with downtime and service disruptions. The impact of service unavailability can be severe, leading to financial losses, reputational damage, and regulatory penalties from the Central bank (RBZ). In addition, regulatory frameworks governing financial services impose stringent compliance requirements that necessitate continuous monitoring of cloud services. This literature review explores the theoretical underpinnings of cloud availability monitoring, examines empirical studies on the subject, identifies emerging global trends, and highlights research gaps that need to be addressed for improved reliability and security in cloud services.

Several high-profile cloud outages have disrupted banking services globally, demonstrating the critical need for robust contingency measures. In December 2021, an AWS outage led to significant disruptions for banks in North America, including major financial institutions like Capital One and Ally Bank. Customers experienced failed transactions, inaccessible banking apps, and delayed fund transfers (Business Insider, 2021). The incident was attributed to a network congestion issue within AWS's US-East-1 region, highlighting the risks of regional cloud dependencies.

In another notable incident, a major Azure outage in March 2022 affected several European banks, leading to widespread service disruptions. Barclays and Deutsche Bank were among those impacted, with customers unable to access online banking portals and perform real-time transactions (Financial Times, 2022). The failure was caused by a configuration error during a routine update, demonstrating the vulnerabilities associated with cloud maintenance operations. In 2023, Google Cloud experienced a severe outage due to a Distributed Denial of

Service (DDoS) attack, affecting major banks in Asia, including DBS Bank and ICICI Bank (Reuters, 2023). Customers reported transaction failures and ATM downtimes, underscoring the cybersecurity risks inherent in cloud-dependent banking.

Near real-time monitoring refers to systems that continuously track web service metrics, such as uptime, response times, and error rates, with minimal delay. Monitoring tools like Prometheus, Nagios, and Datadog are widely used for this purpose, offering alert systems and real-time dashboards (Köhler et al., 2020). Such systems ensure service disruptions are identified promptly, allowing for rapid intervention to minimize downtime. Near real-time monitoring leverages continuous data collection and analysis to track system performance and detect anomalies promptly. Techniques such as flow monitoring, packet sniffing, and telemetry systems have been widely adopted to evaluate endpoint reliability. Tools like Prometheus and Nagios enable real-time data visualization and anomaly detection, ensuring timely intervention in cases of service degradation (Obkio, 2023; IEEE, 2023). Recent advancements in edge and cloud computing have further enhanced monitoring frameworks. By offloading computational tasks to the edge, these systems reduce latency and improve bandwidth utilization, as shown in applications across smart grids and industrial monitoring systems (Zhang et al., 2022).

4.6 Relevant theory of the subject matter

A fundamental theoretical foundation for understanding cloud service availability monitoring is Systems Reliability Theory (SRT), which argues that complex systems must incorporate redundancy and continuous monitoring to ensure high reliability. Cloud infrastructures, being highly complex and distributed, require robust monitoring mechanisms to identify potential failures before they escalate into system-wide disruptions (Barlow et al, 1996). Another relevant framework is Service Availability Theory, which focuses on designing systems that maximize uptime through proactive strategies, including load balancing, automated failover mechanisms, and real-time monitoring (Chen et al, 2018). These theories support the development of cloud monitoring frameworks that utilize real-time analytics and artificial intelligence to predict and mitigate service disruptions before they affect users.

4.6 Empirical studies

Empirical studies highlight the increasing adoption of real-time cloud monitoring solutions in financial institutions. A study by Patel et al. (2020) found that banks and investment firms use cloud monitoring tools such as Amazon CloudWatch, Microsoft Azure Monitor, and Google Cloud Operations to track service performance, detect anomalies, and respond to outages in real time. These tools integrate AI and machine learning algorithms to analyse historical data, predict failures, and enable automated remediation. Kumar and Singh (2021) further examined the role of multi-cloud strategies in enhancing cloud availability for financial institutions. Their findings indicated that institutions that diversified their cloud providers experienced fewer service disruptions compared to those that relied on a single provider. However, the study also identified challenges in maintaining a consistent monitoring framework across multiple cloud environments, given the varying metrics and monitoring capabilities of different providers.

The cloud services industry has been evolving rapidly to improve real-time monitoring capabilities. A global study by Tan et al. (2020) highlighted the regulatory disparities in cloud monitoring practices across regions. The research found that European financial institutions, under strict regulatory frameworks such as the General Data Protection Regulation (GDPR), had more robust monitoring mechanisms than their counterparts in North America and Asia. Additionally, Johnson et al. (2019) found that while cloud providers offer Service Level Agreements (SLAs) with uptime guarantees, the effectiveness of built-in monitoring tools varies significantly depending on workload characteristics and system architecture. This has led to the development of third-party cloud monitoring solutions that offer more granular insights and real-time alerts beyond what cloud providers natively support.

One of the most significant advancements in cloud monitoring is the increasing adoption of AI-driven monitoring systems. Research by Zhang et al. (2022) demonstrated that AI-powered monitoring tools outperform traditional rule-based monitoring in detecting anomalies and reducing response times. These systems leverage real-time log analysis, pattern recognition, and predictive analytics to enhance cloud service availability. Despite their effectiveness, concerns over data privacy, security, and compliance with regulatory standards have slowed down the widespread adoption of AI-driven monitoring in financial institutions. Another emerging trend is the use of edge computing to improve cloud

monitoring efficiency. Traditional cloud monitoring frameworks often rely on centralized processing, which can introduce latency and delay incident response times. Gupta et al. (2023) found that edge computing helps mitigate this challenge by enabling real-time processing closer to the data source, reducing latency, and improving response times for mission-critical financial applications.

Several global studies have also highlighted the persistent challenges in cloud service availability. Miller and Evans (2021) investigated service disruptions across major cloud providers, including AWS, Microsoft Azure, and Google Cloud. Their research found that network failures, misconfigurations, and cyberattacks remain significant causes of cloud outages. While cloud providers invest heavily in redundancy mechanisms, service disruptions still occur, affecting financial institutions that rely on real-time data access for transactions and compliance reporting. Similarly, Nakamura et al. (2022) explored cloud service availability challenges in the Asia-Pacific region. They found that financial institutions in emerging economies face greater risks due to unstable network infrastructure and limited access to advanced monitoring tools. Their research recommended investments in redundancy architectures, AI-driven monitoring, and hybrid cloud strategies to mitigate service disruptions.

Comparative research by Gomez and Torres (2023) examined cloud monitoring adoption across Latin America, North America, and Europe. Their findings indicated that Latin American financial institutions lag in cloud monitoring adoption due to regulatory gaps and limited cloud computing infrastructure. The study called for stronger government regulations to enforce mandatory real-time cloud monitoring frameworks, ensuring financial institutions in the region achieve higher levels of cloud service reliability.

Despite advancements in real-time cloud monitoring, several challenges persist. One major challenge is latency in data processing, where delays in collecting and analysing monitoring data can reduce the effectiveness of real-time response mechanisms. This issue is particularly critical for financial institutions engaged in high-frequency trading, where even milliseconds of delay can lead to significant financial losses. Another issue is false positives and alert fatigue, where AI-driven monitoring systems generate excessive alerts, making it difficult to distinguish between genuine threats and irrelevant notifications. Additionally, security risks associated with real-time monitoring tools arise due to their deep system access, which increases the potential attack surface for cyber threats. Scalability constraints also pose a

challenge, as monitoring systems must efficiently scale alongside expanding cloud infrastructures without introducing performance bottlenecks. Furthermore, compliance with stringent financial industry regulations such as GDPR, PCI-DSS, and Basel III remains a critical concern, as monitoring solutions must align with these requirements while ensuring data security and privacy.

Several research gaps remain unaddressed in the field of near real-time cloud monitoring for financial institutions. Firstly, there is a lack of standardized monitoring frameworks that can be universally adopted across different cloud providers. While some financial institutions have developed proprietary monitoring tools, there is no widely accepted industry standard for realtime cloud service availability monitoring. Secondly, integration challenges in multi-cloud environments need further exploration, particularly in ensuring consistency in monitoring metrics across different cloud platforms. Thirdly, research on the balance between real-time monitoring and regulatory compliance remains limited. Financial institutions must navigate the complexities of data privacy laws while implementing continuous monitoring solutions. Fourthly, while AI-driven monitoring solutions are promising, barriers to adoption, such as cost, complexity, and trust in AI-driven decision-making, require further investigation. Additionally, there is limited research on automated incident response mechanisms that can immediately mitigate cloud service disruptions without requiring manual intervention. Finally, the cost-benefit analysis of deploying real-time monitoring solutions versus traditional monitoring approaches needs further exploration to help financial institutions determine the most effective strategies for cloud service availability monitoring.

According to an industrial study by Damian and colleagues in 2020, it shows that large portion of organisations do not pursue monitoring, and most of these organisations do not have monitoring tools at all, they rely on manual checks such as SSH, grep, and ping. In order to provide insight into the state of industry monitoring practices, their study focused on the following topics: (a) industry monitoring practices and tool adoption; (b) the scope and complexity of industrial monitoring issues; and (c) the function of software architecture and software processes with regard to monitoring strategies. They use interviews and a web survey with more than 140 practitioners from more than 70 organizations as part of their mixedmethods empirical research. The following were the main conclusions.(a) Despite the fact that the downtime of their applications is closely correlated with the automation and responsiveness that monitoring enables, industrial decision-makers do not view monitoring as a critical asset; (b) monitoring is carried out using antiquated technology, primarily MySQL

querying or something similar (e.g., Nagios); and (c) clients, not application owners, are the ones who find incidents. (Damian et al., 2020). These results demonstrated that only a small number of large organizations are actively monitoring their services because of the aforementioned issues, even though there are many monitoring tools available on the market. According to research, in order to prevent service interruptions and harm to one's reputation, monitoring frameworks that are simple to implement must be developed. Additionally, according to recent studies, 50% of organizations lack monitoring standards, and some of the available tools are difficult to use.

For Nucleic Acids Researchers in Berlin, a related system called Aviator—a web service for tracking cloud service availability—was created in 2021. It was employed to keep an eye on the availability of the instruments they needed for their study. They set up a crawler that attempts to access the webpage twice a day in order to test a web server's availability. In a Docker container, they used pyvirtualdisplay to display a Python script that was running Selenium with Google Chrome. Their implementation waits up to 30 seconds when a webpage is queried, and another 30 seconds if status code 202 is returned, which happens often when R Shiny servers are launched.. The JavaScript console is used to extract performance metrics like memory usage and response times for the frontend and backend. The ChromeDriver log is used to extract additional data, like the quantity of requests, the SSL certificate status, or the response status code. After that, the web server implementation receives the gathered data and updates the database appropriately. Only when their response return code is 200 are they deemed to be online. A website was considered online for the day if it was accessible in at least one of the two ways at least once every day. Nonetheless, there is a significant reliance on web services nowadays, and multimillion-dollar transactions are carried out via these websites. The business can suffer significant losses if a service is unavailable for a few hours; therefore, polling frequency should be increased to minimize extended outages.

During the COVID era, Aviator was used to track services related to COVID-19. Their goal was to evaluate how well it performed in dynamic settings. The findings show that there were issues with false positives in this app. Determining whether a reachable website is also functional is an open question that the researchers neglected to address. An automatic link availability checker could be incorporated as a first step to improve the relationship between a website's functionality and availability. Then, a buildup of broken links might serve as a clue that something is broken. Similarly, a collection of site assets that don't load could be a

sign of a problem. Monitoring the modifications made to the web pages and queries over time may also yield more information (Tobias et al., 2021).

Another study was conducted in 2021 at Ryerson University in Toronto, Canada. A team of researchers investigated the difficulties in cloud service monitoring. Resource intensity, scaling difficulties, monitoring requirement dynamics, and security concerns were some of these difficulties (William et al., 2021). By avoiding storing repetitive values, which can reduce the size of stored data by up to 80%, building a fault-tolerance monitoring system based on clustered architecture, and combining role-based monitoring templates with agent-based monitoring, my proposed system aims to address some of these issues. It also uses an event processing engine to refine the data collected and provide a reliable and comprehensive monitoring solution. Furthermore, a thorough framework covering PaaS, SaaS, and IaaS will be part of the suggested solution. To further improve security, data exchange between the service, monitoring tool, and databases is always encrypted.

According to recent research, maintaining the effectiveness, dependability, and performance of cloud services requires constant monitoring. However, the industry's current ad hoc monitoring methods may lead to inconsistencies and inefficiencies (Srinivas et al, 2024). An empirical evaluation on "Intelligent Monitoring Framework for Cloud Services: A Data-Driven Approach" found that, in spite of great efforts to guarantee reliability, production incidents or failures are unavoidable, can harm customers, and necessitate a large amount of engineering resources and manual labor to mitigate. Therefore, minimizing the impact on customers and lowering the related expenses require the early detection and mitigation of incidents.

Service providers proactively identify and mitigate incidents before they have an impact on customers and continuously monitor service health in order to address this. Service owners add monitors based on their knowledge of service architecture, key dependencies, and service-level agreements in the current trial-and-error method of creating monitors. They also emphasized that the way services are currently monitored is error-based, with service owners adding monitors in accordance with their knowledge of the service architecture, key dependencies, and service-level agreements. The majority of monitoring tools nowadays are redundant monitors, which leads to noisy alerts and wasted effort, according to the results. In light of these problems, I present a monitoring framework that is universally applicable

regardless of resource classes. It should also prevent noisy notifications by implementing a snoozing mechanism in alerting until the problem has been fixed.

As a relatively new technology, there is currently no widespread agreement on suitable evaluation standards for cloud computing and its monitoring tools. Appropriate monitoring tools are desirable, and their usefulness and effectiveness should have been assessed. Below is a discussion of several empirical studies. MoDe4SLA, as proposed by Bodenstaff et al. (2011), enables the monitoring and management of inter-service dependencies within a composition. They conducted an experiment with 34 participants to empirically validate their methodology.

By asking specialists to oversee simulated service composition executions using MoDe4SLA, the authors assessed usefulness. But instead of keeping an eye on the quality of the services, they concentrated on the outcomes, which produced a well-rounded set of services.

A framework for service management based on SLAs was proposed by the SLA@SOI project. They provided assessments that showed how SLA@SOI was applicable. Additionally, they provided a case study on how the SLA@SOI framework is applied to eGovernment domains. Instead, the case study served as a proof-of-concept, and the findings ignored the monitoring approach evaluation in favor of concentrating on the full framework. CaSViD was introduced by Emeakaroha et al. (2012), who also used a proof-of-concept to assess their proposal. They assessed two aspects: (i) the architecture's ability to automatically determine the effective measurement interval for efficient monitoring, and (ii) its ability to monitor applications at runtime in order to detect SLA violations. The authors did not concentrate on how users perceived their solution, and the evaluation was not rigorous.

Lastly, a number of publications discuss their experiences using monitoring tools to assess performance, latency, efficiency, and other low-level NFRs. GMonE (Global Monitoring System), a cloud monitoring tool, was proposed by Montes et al. (2013). They used an experimental testbed to assess GMonE's overhead, scalability, and performance. High performance, low overhead, scalability, and elasticity were all tested in a large-scale cloud environment. In-depth tests were conducted by Meng et al. (2013) in a cloud environment simulation that included real-world system and network traces. The findings demonstrate that their method outperforms others in terms of multi-tenancy performance, scalability, and monitoring costs.

However, real environments and users who could offer feedback and help improve the approach with their perspectives were not included in their evaluation. Some limitations in the empirical evaluation of cloud monitoring solutions have been identified through the analysis of the aforementioned studies. These include: (1) the dearth of empirical studies evaluating the experience of users with the monitoring tool; (2) the absence of studies analyzing the interaction between the monitoring solution and its users to define the quality characteristics to be monitored; and (3) the analysis of the likelihood of intention to use a particular solution when users need to monitor their cloud services. Therefore, when a request is posted to the system, my suggested framework will record the number of failed jobs and response times, enabling us to see the system from the viewpoint of the user. In most cases, this enables WS providers to make the required modifications to ensure that their clients receive top-notch service.

Cloud services must be monitored in almost real-time to guarantee availability and dependability. The efficiency of these systems will be further increased by addressing issues like complexity, security, false positives, and redundant, noisy alerts. Organizations will be able to easily monitor their services with the help of the suggested system's straightforward architecture. Because the system monitors the system from the perspective of its users, adoption of this framework will increase the number of users who can monitor their services and improve the quality of service.

According to the literature, in order to improve service availability, financial institutions and the cloud service sector are actively investing in near real-time monitoring solutions. A theoretical basis for comprehending the significance of ongoing monitoring in intricate cloud environments is provided by the Systems Reliability Theory and the Service Availability Theory. Empirical research highlights the benefits of edge computing, multi-cloud strategies, and AI-driven monitoring in guaranteeing the dependability of cloud services. There are still a lot of unanswered questions, though, especially in the areas of cost analysis, automated response systems, compliance, standardization, multi-cloud integration, and AI adoption. In the financial industry, where even the smallest disruption can have serious repercussions, closing these gaps is essential to guaranteeing reliable, safe, and compliant cloud services.

2.3 Chapter Summary

2.3.1 Major Findings

1. Incidents are being discovered by users rather than service owners
2. 18% of companies have no monitoring tools; they rely on manual checks, such as SSH, grep, and ping
3. Financial institutions in developing countries are more subject to cloud service outages due to unstable network infrastructure
4. Network failures, misconfigurations, and cyber-attacks are the most common causes of service disruptions in the financial industry
5. Diversity in cloud providers can reduce downtimes
6. Countries with strict regulations offer better proactive monitoring of cloud services
7. SLAs encourage the development of effective monitoring tools
8. Edge computing reduces disruptions associated with centralized processing
9. Globally, financial institutions are using AI and ML tools for monitoring, but their adoption has slowed down due to security and privacy concerns.

2.3.2 Research Gaps

1. Develop universally accepted frameworks that can be adopted across different cloud platforms.
2. Addressed the challenges associated with monitoring across multiple cloud providers in real time.
3. How can financial institutions balance the need for real-time monitoring with stringent data protection laws such as GDPR and PCI-DSS?
4. How to overcome the barriers to AI adoption in cloud monitoring?
5. Lack of research on automating mitigation strategies in real-time cloud service disruptions.
6. More research is needed to determine the financial implications of deploying real-time monitoring solutions versus traditional periodic monitoring approaches.

2.3.3 Notable challenges

1. False positives
2. Excessive alerts
3. Maintaining a constant framework for different cloud environments
4. Privacy and security risks
5. Limited scalability in monitoring tools
6. Complex architectures that are not user-friendly
7. Balance between monitoring and industry regulations

2.3.4 How my research addresses some of the challenges

1. Snoozing of alerts
2. Post a request and validate to avoid false positives
3. User-friendly dashboard to avoid complexity
4. Use data encryption to combat data security and privacy issues

2.4 Conclusion

This chapter has explored the current state of cloud service monitoring in financial institutions, particularly in developing countries, and highlighted significant challenges and gaps in the field. Key findings indicate that many incidents are still discovered by end-users rather than service providers, with a concerning 18% of companies lacking formal monitoring tools. Cloud outages in developing regions are largely due to unreliable infrastructure, and the primary causes of service disruptions include network failures, misconfigurations, and cyberattacks. However, diversification of cloud providers, strict regulatory environments, and the integration of technologies like edge computing and AI have been identified as potential solutions to improve monitoring efficacy and reduce downtime.

The research also identifies pressing gaps, such as the need for standardized monitoring frameworks, real-time multi-cloud monitoring solutions, and strategies to overcome AI adoption hurdles. Furthermore, it draws attention to unresolved challenges, including false positives, alert fatigue, and the difficulty of maintaining consistency across varied cloud architectures while remaining compliant with industry regulations.

To address some of these challenges, this research proposes practical solutions such as implementing alert snoozing mechanisms, using validation techniques to reduce false positives, designing intuitive dashboards to enhance usability, and employing encryption to mitigate privacy and security concerns. These contributions aim to support the development of more resilient, scalable, and secure cloud monitoring practices tailored to the unique needs of financial institutions operating in diverse regulatory and infrastructural environments.

Chapter 3: Research Methodology

Brief description

The cloud services that CABS provides to its stakeholders will be the subject of the study. WS Monitor, a Laravel web application designed to track and monitor the availability of these services, will be used in this study. The OS's scheduled task for the application will cause web services to be polled to assess their health, including availability, response times, and other metrics. It will send requests on a regular basis and check for answers using the architecture of SOAP and RESTful APIs. The statistics are saved in the database and log for reporting purposes, and an alert is meant to be generated in the form of a text message each time we are unable to contact a service or a TC Server. The researcher has created seeders and database migrations for the database, which make it simple to create tables and add initial values to them, respectively. To determine whether the method is reliable in identifying anomalies in the event of incidents in a timely manner, recorded responses will be statistically evaluated. The Laravel PHP framework is used in the development of the system.

- The study will be conducted on:4 TC Server
- 4 API ➤ 11 Webservices

3.1Research Design

A comprehensive approach used to integrate various components of the research study in a way that will aid in successfully resolving the research problem is referred to as research design. A reference regarding the collection, measurement, and analysis of data is also included in the research design (Johnson et al, 2020). Since hypothesis testing is the foundation of the research, an experimental study will be used as the research design. The researcher constructed a prototype of the real system in order to carry out the experimental investigation. Efficiency and dependability will be evaluated using the prototype.

3.1.1 System Development Methodology

The set of procedures used to conduct research is known as the system development methodology. Software development work is divided into discrete phases by a software development methodology (Ponto, 2021). The evolutionary prototyping model was used to design the system. Below is a more thorough explanation of the evolutionary prototyping model.

A working software model with a few restricted features is called a prototype. The prototype is an additional effort that should be considered when estimating effort because it does not always contain the exact logic used in the final software application. Users can assess developer proposals and test them out prior to implementation through the use of prototyping. It also aids in comprehending the requirements, which are unique to each user and might not have been considered by the developer when designing the product. In this study, WS Monitor was first created to monitor the availability of just three cloud services: a web service, an API, and a test full server. The evolutionary prototyping for this work is depicted in the diagram below, Figure 1:

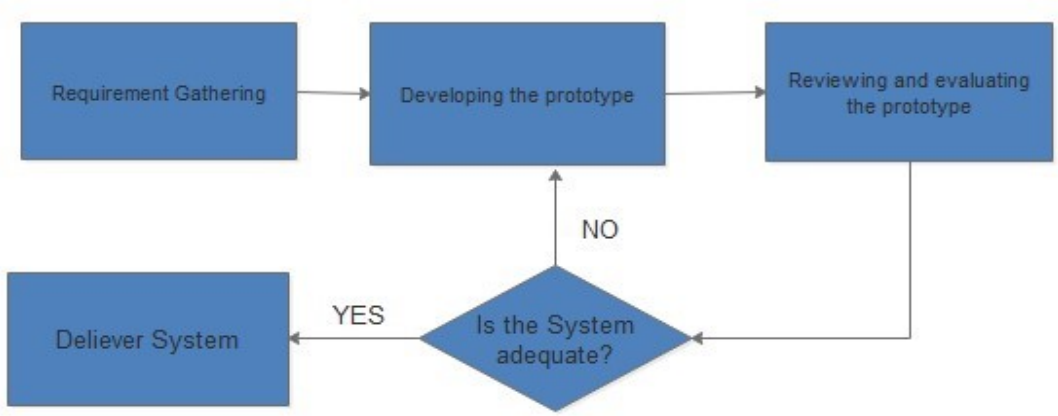


Figure 1: Evolutionary prototyping

Basic Steps for Prototyping

Requirement gathering

- All of the fundamental needs and specifications for the prototype are gathered at this first stage of evolutionary prototyping. At this point, working with the Bank and the ICT department, functional and non-functional requirements are established.

Developing the initial prototype

- Following requirements collection, a prototype is created based on user-specified requirements; therefore, the prototype should be able to conduct polling, save responses to the database and log, and produce alerts. These features provide a

hypothetical image of the finished system, even though they might not be precisely like the one that will be developed.

Reviewing and Evaluation of the prototype

- A few project stakeholders are shown the prototype. In order for the monitoring tool to address all of the client's concerns, the feedback from the presentation is extremely important because it is incorporated into the prototype that is being developed. ***Revising and Enhancing the prototype***
- The prototype that is being developed is improved using the findings from the evaluation. At this point, the prototype is modified until all project stakeholders are happy. The system is delivered once all requirements have been satisfied; if not, the prototype is improved and revised once more until we satisfy the bank's requirements.

3.1.2 Functional requirements

- Check health every two minutes
- Snooze alerts to avoid noise alerts
- Keep summary of responses in a log and database
- Visualize current status of services health
- Encrypt the database to guarantee data security and privacy
- Detect when a service goes down immediately and send SMS alert

3.1.3 Non-Functional requirements

- Easy to understand
- Informative
- Quickness
- Easy to add other endpoints that needs monitoring
- Have open windows for further development

3.1.4 Tools used

Software requirements

- Apache v2.4.54.2
- MySQL v8.0.31

- Composer 2.8.4
- Windows 10 OS
- VS Code V1.77.3

Other software requirements are stipulated in the code snippet below:

```
"require": {
        "php": "^8.0",
        "artisaninweb/laravel-soap": "0.3.0.10",
        "codedredd/laravel-soap": "v2.0.0-alpha.0",
        "fideloper/proxy": "^4.4",
        "fruitcake/laravel-cors": "^2.0",
        "guzzlehttp/guzzle": "^7.0.1",
        "laravel/framework": "^8.0",
        "laravel/passport": "^10.4",
        "laravel/tinker": "^2.5"
    },
    "require-dev": {
        "facade/ignition": "^2.5",
        "fakerphp/faker": "^1.9.1",
        "laravel/sail": "^1.0.1",
        "mockery/mockery": "^1.4.2",
        "nunomaduro/collision": "^5.0",
        "phpunit/phpunit": "^9.3.3"
    },
```

Hardware requirements

- Ram 8GB or more
- Processor 2.60GHz 2.59 GHz

3.1.5 Data flow diagram

Figure 2 illustrates the data flow of the proposed monitoring system. The system periodically initiates polling tasks via scheduled operations on the host computer. Depending on the service type, it performs different checks: an HTTP GET request for APIs, an HTTP POST request for web services, and a socket connection attempt for TC Servers. The results of these checks are stored in a database. If any service is found to be unresponsive or down, an SMS

alert is immediately generated. In parallel, the system also verifies whether previously recorded downtimes have ended. For instance, if a server has been marked as down for the past 20 minutes and the current polling cycle detects that it is now operational, the system updates the server’s status to "UP." This mechanism ensures accurate tracking of both active downtimes and service recoveries

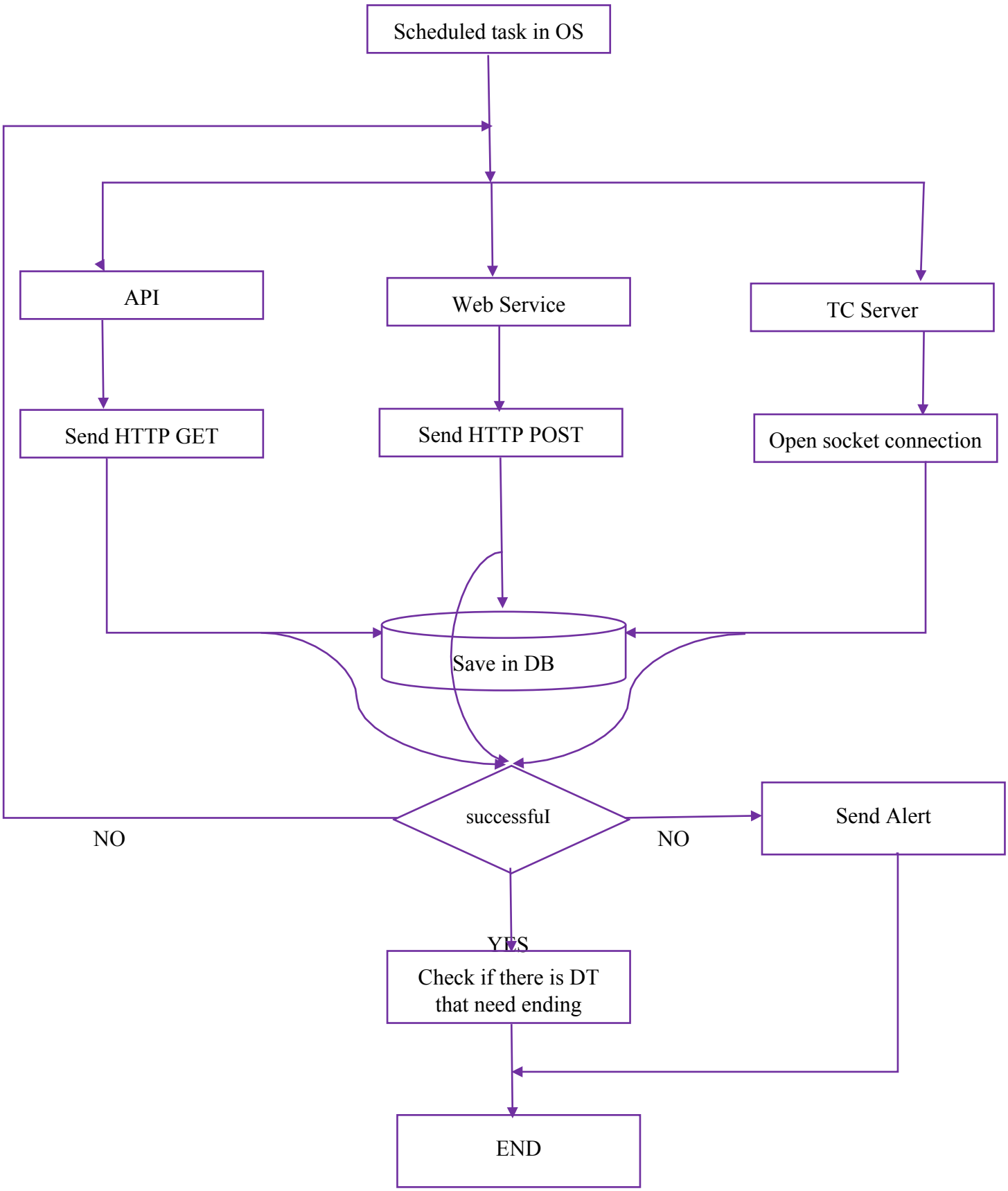


Figure 2: Proposed system dataflow diagram

There is a scheduled task in the OS which is triggered every two minutes to poll web services to check health of web services, APIs and TC Server. The diagram, figure 3, shows the task which triggers polling after every 2 minutes:

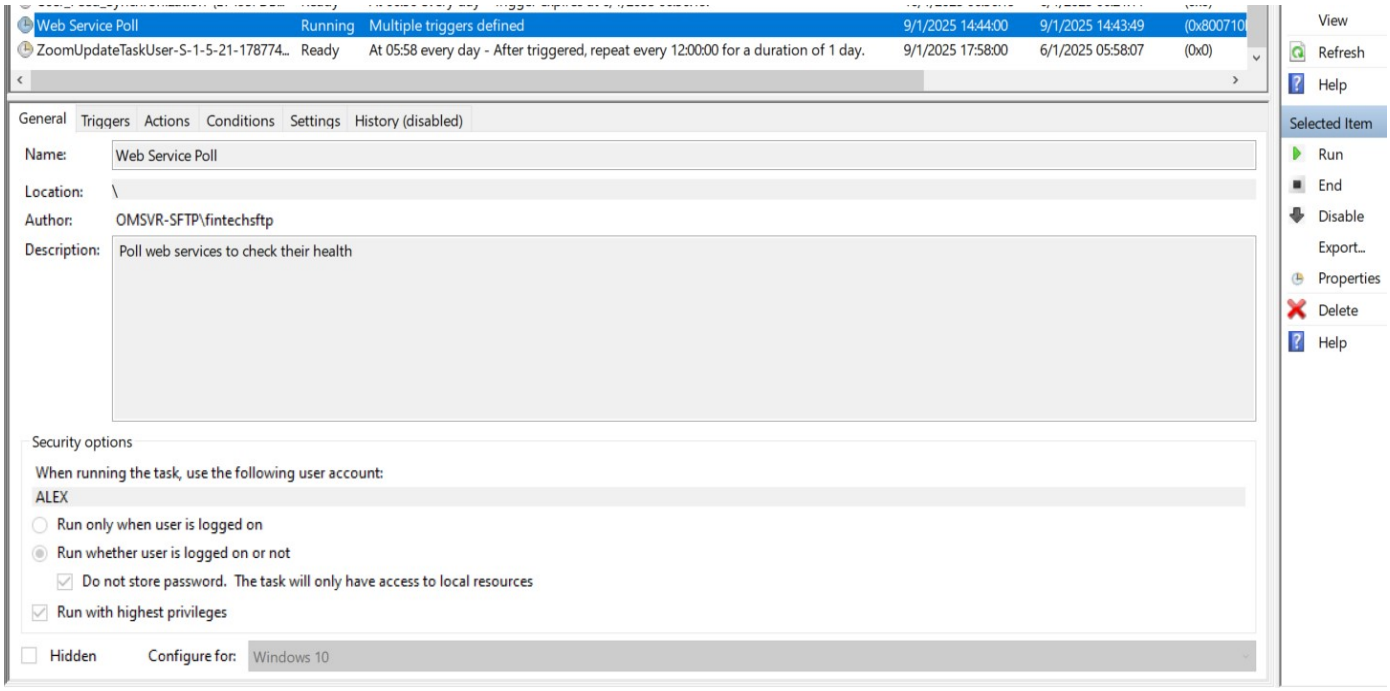


Figure 3: Scheduled task in OS

Every two minutes *Cloud services Poll* is executed and the responses of the polling are saved and evaluated so that the system can trigger an alert if necessary. On APIs, we send a GET request to perform health checks on the API, verifying its availability and responsiveness. This helps us to get a clue about API performance, error rates, and we can verify its consistency across different endpoints over time.

For web services, we use a POST request to send data to the server for processing. Since data processing can involve accessing sensitive parts of our system, POST requests are more secure as they don't expose sensitive data in the URL. The request sends sample data for processing to verify that the web service can process it correctly, which helps to reduce false positive hits. If we get a correct response, it means the service is up; else it will be down, hence the system generates an SMS alert. Post request allows for more comprehensive testing of cloud services as it can simulate real-world user interactions and data processing. We can also detect errors and exceptions that might not be caught by simple health checks, such as

pinging or sending a GET request. Monitoring at this level gives us an overview of what our clients are experiencing, which is the overall aim of this research: to enhance service delivery. POST request will simulate a more realistic load on cloud services, helping to identify performance bottlenecks and scalability issues.

For the TC Server, the WS Monitor initiates a socket connection to the TC Server by specifying the server IP address and port number. The TC Server accepts the connection request, and a socket connection is established. The WS analyses the responses from the TC Server to monitor its performance, health, and other relevant metrics like response times.

3.1.6 User interface design

Figure 4 depicts the appearance of a front-end visualization dashboard that is used by the Operations team on duty to monitor cloud services. It shows whether a service is up or connected for TC Servers, as well as the response times for cloud services. The colour codes help to quickly identify services that are down. Green colour reflects that something is up or connected, and red tells us that something is down or not connected.

Figure 4: User Interface visualization dashboard

For example, from the diagram shows that all other services are down, with the EQUALS API the only one which is up, with a 10.78s response time. This type of visualization helps the OPS team to quickly asses performance of services and identify incidents. If something happens with the SMS Alerting service, we can still have meaningful insights from this dashboard.

3.1.7 Use Case Diagram

The diagram below, in Figure 5, is a Use Case Diagram for a *Cloud Services Monitoring* system. It illustrates the interactions between two user roles, Admin and OPS (Operations), and the key functionalities of the system.

- Actors:
  - Admin: Responsible for configuration and management tasks.
  - OPS: Handles operational monitoring and response tasks.
- Use Cases (System Functions):
  - Login: Both Admin and OPS users authenticate to access the system.
  - Polling: Periodic checks are initiated to monitor the status of cloud services.
  - Save status: Monitoring results are saved for logging and analysis.
  - Send alerts: Notifications (e.g., SMS or email) are sent when issues are detected.
  - Receive alerts: OPS receive alerts about service issues.
  - Respond to the issues: OPS take appropriate actions to resolve detected issues.
  - Add endpoints: Admin can add new cloud service endpoints to be monitored.
  - Visualize: Users can view monitoring results and system status through dashboards or reports.

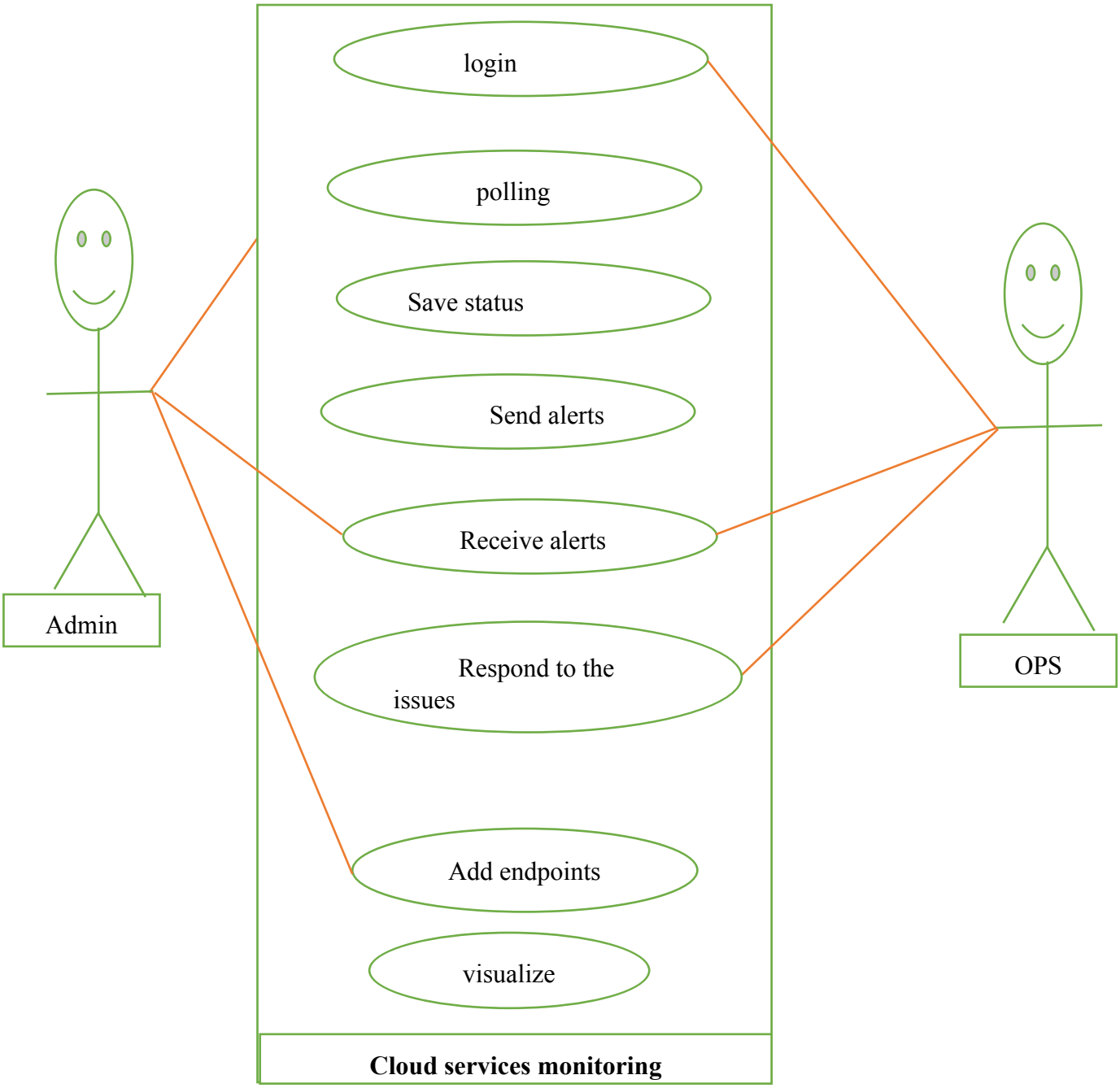


Figure 5: Use case diagram

3.2Data Collection Approaches

The researcher used active data collection methods. Active monitoring is a technique where a Monitoring tool actively sends requests or synthetic transactions to a web service to assess its availability, performance, and reliability. Unlike passive monitoring (which collects data from logs and real user activity), active monitoring proactively tests services, even when

there are no real users. Synthetic API monitoring involves actively testing APIs by sending simulated requests at scheduled intervals. The collected data helps measure API availability, performance, and reliability. Key data collection methods include:

Response Time Metrics Collection measures the total time taken for an API request to complete. Breakdown includes TCP Connection Time (Time to establish a network connection), TLS Handshake Time (Time taken for SSL/TLS negotiation (if HTTPS)), First Byte Time (Time until the first byte of response is received), Total Response Time (Time until the complete response is received).

```
public function testTCServer(){
        //Log::info("TC Server (.22) start");
        $server = 'TC Server (.22)';
        $data = array();

        try{
            $socket = @fsockopen("180.10.7.22", 7996, $errno, $errstr);

            if($socket){
                $data['status'] = "connected";

                //Check if there is a downtime that needs to be ended
                $this->downtimeEnd($server);

                return json_encode($data);
            }
else{
                $data['status'] = "not connected";
                $data['success'] = "failed to reach TC Server";
                $data['responseTime'] = 0;
                $data['messages'] = "server could not be reached";
                $data['status'] = 0;
                $status = 0;
```

From the above code snippet, we try to open a socket connection with server .22 to check its availability, then our responseTime variable will keep track of the time taken to establish a connection with the server. If the connection is successful, then we conclude that the server is up.

HTTP Response Status Code Tracking

This is the process of monitoring and analysing the **HTTP status codes** returned by a web server or API in response to client requests. This tracking helps determine the availability, performance, and reliability of cloud services. Tracking these codes allows organizations to quickly detect problems, such as unauthorized access attempts, API rate limits, or backend service failures. So here we send Http::get() to monitor an API endpoint and send alerts if it is unreachable.

```
try{
                $response = Http::get('http://180.10.0.117:10011/hello');

                if(!isset($response)){
                    $this->sendAlerts(null, $response->status(), "Request Timed
Out", "N/A", null, null, $system);
                }
else{
                    if($response->status() != 200){

                        $this->sendAlerts($system, $response->status(),
"Unreachable", "N/A", null, null, true);
                        return response()->json(['status' => 'Down']);
                    }
```

For example, in the above snippet, we are trying to get ('http://180.10.0.117:10011/hello'). If the response code is not 200, then we send an alert, meaning the request was not successful, and the service status is deemed as down.

Both Response Time Metrics Collection and HTTP Response Status Code Tracking are essential components of synthetic API monitoring. While response time analysis helps optimize API performance, status code tracking ensures service reliability by detecting failures.

3.3Population and Sample

A population sample is a subset of individuals, items, or data points selected from a larger group for research, analysis, or statistical study. The experiments were carried out on CABS web services. Specifically, the experiment had 4 APIs, 4 TC servers, and 11 Webservices. These services were actively monitored for 60 days using WS Monitor from the first November 2024, and the results were recorded for statistical evaluation to determine the reliability of an exposed high-level endpoint for monitoring of cloud services.

3.4Research Instruments

The research was carried out using WS Monitor, which is a research instrument developed by the researcher. This system can monitor cloud services, including APIs and TC Servers. The system offers near real-time monitoring of services by doing response time monitoring and response code tracking to give an overview of the cloud services' health.

3.5Data Analysis Procedures to be used

The data analysis procedure helps identify trends, detect anomalies, diagnose failures, and improve the efficiency of our monitoring tool. Below is a structured data analysis procedure for WS Monitor using synthetic monitoring.

1) Collect Monitoring Data
    - The tool generates data at regular intervals by sending simulated requests to APIs.
2) Data Cleaning & Transformation
    - Before analysis, the collected data needs to be formatted and cleaned to remove inconsistencies.
3) Statistical & Trend Analysis
    - Statistical analysis helps identify normal vs. abnormal patterns in API performance.
4) Compute Key Performance Indicators (KPIs)
5) Data Visualization & Reporting
    - Visualizing data helps teams understand performance trends and act.

A structured data analysis procedure in synthetic API monitoring ensures that organizations can detect performance bottlenecks, diagnose failures, and optimize API performance. By leveraging statistical methods, anomaly detection techniques, and visualization tools, teams can proactively enhance API reliability, reduce downtime, and improve user experience.

Chapter 4: Data Presentation, Analysis, and Interpretation

Introduction

This chapter covers a comprehensive analysis of the performance data collected by WS Monitor, a cloud service monitoring application, over a period of eight weeks. The monitoring app, which is hosted locally but designed to track the uptime and availability of cloud-based services, was evaluated based on its ability to accurately detect service downtimes. The data includes key metrics for each week, which are True Positives (TP): Downtimes the app correctly identified, False Negatives (FN): Downtimes that occurred but were not detected, and False Positives (FP): Downtimes falsely reported. These values are used to assess the effectiveness of the monitoring solution in identifying incidents that impact cloud service availability. To evaluate the performance, key statistical metrics such as detection rate (recall), miss rate, precision, and F1 score are calculated. The results provide insights into the app's reliability and highlight areas for improvement. This analysis also explores potential reasons for missed detections and suggests strategic enhancements to improve monitoring accuracy, especially considering the limitations of hosting the monitoring system locally.

4.6 Analysis and interpretation of results

Real-time SMS Alert

The screenshot below, Figure 6, shows a real-time SMS alert generated when a service goes down. For IT operators, these notifications act like an early warning system; the sooner they know something is wrong, the faster they can jump in to fix it. Unlike emails or dashboard alerts, SMS messages cut through the noise because they're instant, direct, and hard to miss, even if you are away from your desk or working off-hours. In the world of cloud services, downtime does not just mean inconvenience; it can translate into lost revenue, broken user trust, and even breach of service agreements. An SMS alert ensures that operators are immediately aware of a problem before it spirals into something bigger. It gives them the chance to start troubleshooting right away, mobilize support teams if needed, or at least communicate proactively with users. In short, service down SMS alerts keep operators in control, minimize damage, and help maintain the reliability and reputation of the cloud services they're trusted to manage.

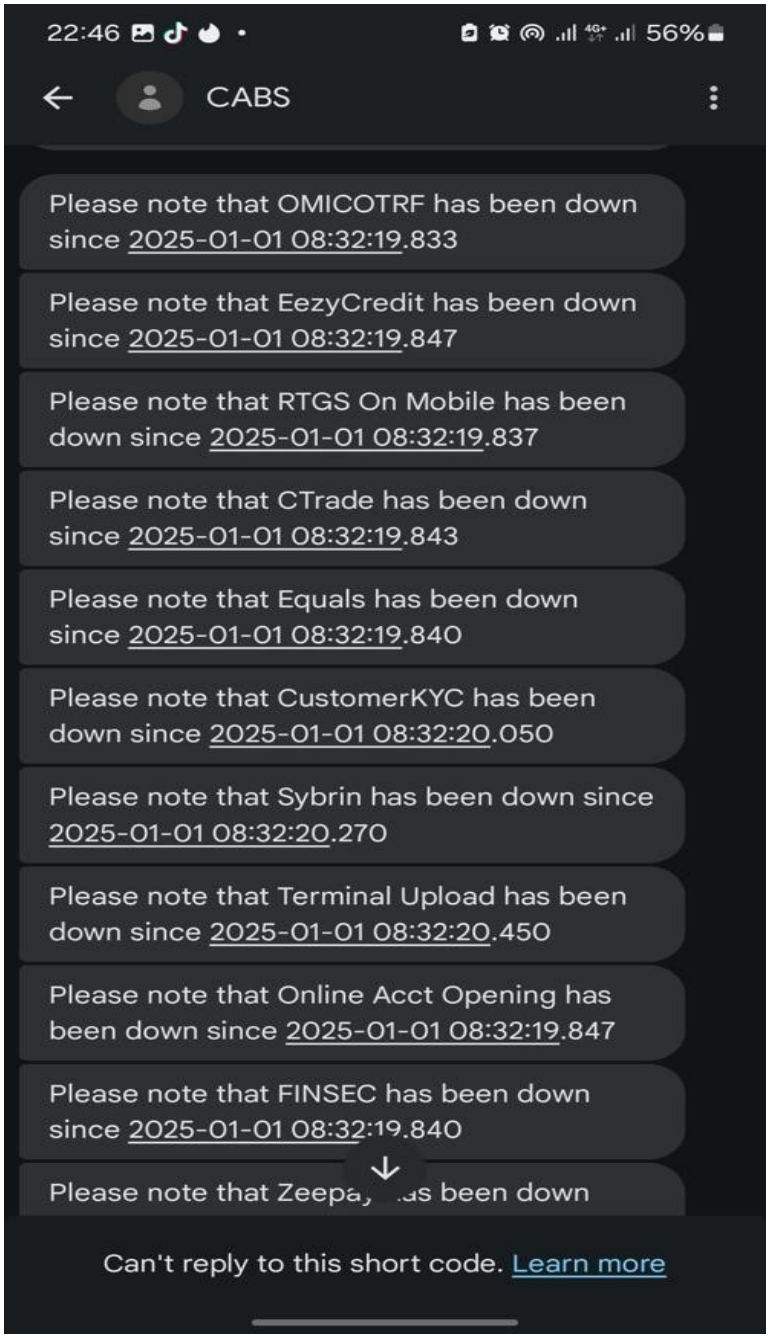


Figure 6: SMS ALERTS generated

Data Encryption

Figure 7 shows that the researcher uses Laravel's `Crypt::encryptString()` method to encrypt sensitive data before storing it. This method is built on top of a well-respected encryption standard, AES-256-CBC, which Laravel implements via PHP's OpenSSL extension. Advanced Encryption Standard is one of the most trusted encryption algorithms in the world. The "256" in AES-256 refers to the length of the encryption key, 256 bits, which

provides a very high level of security. The "CBC" part stands for Cipher Block Chaining, a mode of operation that enhances encryption strength by using what's called an initialization vector (IV). Basically, this makes sure that even if you encrypt the same string multiple times, the output will be different each time, which makes it much harder for an attacker to detect patterns or guess the original input. In summary, `Crypt::encryptString()` gives the application a secure, developer-friendly way to protect sensitive information using a proven encryption algorithm.

```php
public function run()
{
    //
    Auth::create([
        'user' => Crypt::encryptString("ZSST2401"),
        'pass' => Crypt::encryptString("Black@2019"),
    ]);

    Auth::create([
        'user' => Crypt::encryptString("TEMENOS05"),
        'pass' => Crypt::encryptString("Nash@cabs?01"),
    ]);

    Auth::create([
        'user' => Crypt::encryptString("SYBRINT2401"),
        'pass' => Crypt::encryptString("Bin@1212"),
    ]);

    Auth::create([
        'user' => Crypt::encryptString("ZEEPAY01"),
        'pass' => Crypt::encryptString("React@2021"),
    ]);

    Auth::create([
        'user' => Crypt::encryptString("TEMENOS02"),
        'pass' => Crypt::encryptString("Nash@cabs?01"),
    ]);
}
```

Figure 7: Proof of the Encryption standard used

Snoozing of Alerts

The code snippet below, Figure 8, plays a critical role in enhancing the user experience of a cloud service monitoring application by implementing a "snooze" mechanism for alerts. When a service goes down, the system avoids repeatedly notifying users every few minutes, which could become overwhelming and counterproductive. Instead, it checks the time since the last alert and only sends another one if at least 10 minutes have passed. This helps ensure that notifications remain meaningful and actionable, rather than turning into background noise that users might start ignoring. By spacing out alerts intelligently, the system strikes a balance between being responsive and being respectful of the user's attention, especially during ongoing incidents that take time to resolve.

```
else{
    //Log::debug("Downtime record found.");
    //Send alert after 10 minutes
    $downtimeRec = Downtime::where('service', $service)
                    ->where('end', null)
                    ->orderBy('id', 'desc')
                    ->first();

    $start = new DateTime($downtimeRec->start);
    $last = new DateTime($downtimeRec->last_notification);
    $now = new DateTime(now());

    $since_start = $last->diff($now)->format('%i');
    //Log::debug("Lapsed: " . $since_start);
    //Send alert if 10 minutes lapsed since last notification
    if($since_start >= 10){
        //Log::debug("10 minutes lapsed since last notification.");
        try{
            //Update last notification timestamp first to avoid sending of multiple alerts
            $downtimeRec->last_notification = $now;
            $downtimeRec->save();
        }
        catch(Exception $ex){
            //Log::debug("Failed to update record");
        }
```

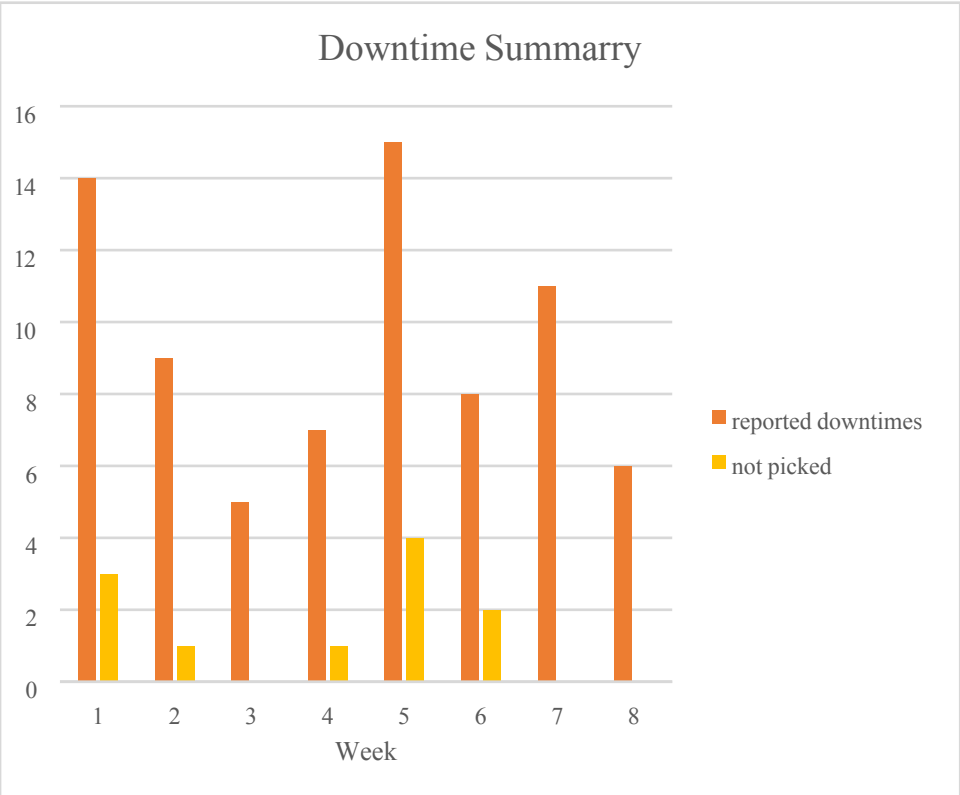
Figure 8: Proof of snoozing ALERTS

4.2 Statistical Evaluation

Week	**Reported Downtimes (Total Incidents)**	**FALSE NEGATIVES**	**TRUE POSITIVES**
Week 1	14	3	11
Week 2	9	1	8
Week 3	5	0	5
Week 4	7	1	6
Week 5	15	4	11

Week 6	8	2	6
Week 7	11	0	11
Week 8	6	0	6
Total	75	11	64

NB: No false alerts were reported.



1. Detection Rate (Recall)

Formula:

$$\text{Recall} = \frac{\text{Picked}}{\text{Total Downtimes}} = \frac{64}{75} \approx \boxed{0.853} \text{ or } \boxed{85.3\%}$$

This means the app successfully detected about **85%** of all downtime incidents.

Figure 9: Recall calculation

❌ **2. Miss Rate**

Formula:

$$\text{Miss Rate} = \frac{\text{Not Picked}}{\text{Total Downtimes}} = \frac{11}{75} \approx \boxed{0.147} \text{ or } \boxed{14.7\%}$$

> So, approximately **15%** of downtime events were **missed** by the app.

Figure 10: Miss rate calculation

F1 Score Calculation

Step 1: Precision

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{64}{64 + 0} = 1.00$$

Step 2: Recall

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{64}{64 + 11} = \frac{64}{75} \approx 0.853$$

Step 3: F1 Score

$$\text{F1} = 2 \times \frac{(\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} = 2 \times \frac{(1.00 \times 0.853)}{1.00 + 0.853}$$

$$\text{F1} \approx 2 \times \frac{0.853}{1.853} \approx 0.921$$

✅ **F1 Score ≈ 0.921 (or 92.1%)**

Figure 11: Precision and F1 Score calculation

An F1 score of 92.1% indicates that the monitoring app is very effective at detecting real downtimes with high precision, no false alarms, and strong recall; most real issues were caught. The only area for improvement is increasing recall, ensuring fewer downtimes go undetected.

- **Precision: 100%** – indicating that all reported incidents were accurate (no false positives).
- **Recall: ~85.3%** – showing that most, but not all, actual downtimes were successfully detected.
- **F1 Score: ~92.1%** – a balanced measure reflecting both precision and recall.

4.3 Interpretation

- Overall performance

  The app performs reasonably well, detecting the majority of incidents (85%). In many real-world applications, anything above 80% detection is a decent benchmark, depending on the criticality of the system.

- Best Weeks

  Weeks 3, 7, and 8 had 100% detection rates, showing that under ideal conditions, the app can be fully effective.

- Worst Week

  Week 5 had the highest number of missed downtimes (4 out of 15), with only about 73% detected that week.

- Reliability observation

  The app is mostly consistent, but there are occasional drops in performance. This indicates external factors or intermittent app issues, like power cuts and network issues, may be affecting reliability.

  The app is doing a solid job detecting cloud service downtimes with an 85.3% success rate. Depending on its criticality, this is a significantly positive result. The fact that it performs perfectly some weeks suggests the core logic is sound, but enhancements in infrastructure robustness**,** detection granularity**,** and redundancy can help close the gap.

4.4 Possible reasons for missing downtimes

1. The app is cloud-based but hosted on a local server; it may miss external events during local outages.
2. The app may not have visibility into all service endpoints or might lack required monitoring privileges, especially those cloud services with external integration with other service providers.
3. Misconfigured agents might not report events properly.
4. If thresholds for identifying a “downtime” are too loose, for example, ignoring spikes or latency, true downtimes may be skipped.
5. Insufficient monitoring intervals, might consider reducing polling intervals from 2 to 1 minute.

4.5 Suggestions to improve monitoring App Performance

- Increase polling frequency, which means lowering the interval between checks to catch short-lived incidents.
- Host the app across multiple servers or cloud providers to avoid a single-point failure.
- Add third-party or external monitoring services that run independently to verify incidents.
- Ensure all critical components are being actively monitored, for example, databases, APIs, and auth services
- Implement machine learning or statistical anomaly detection for dynamic thresholds.
- Review logs from missed downtimes to find out why they weren't picked.
- Compare detected downtimes against service-level agreements to quantify gaps.

4.6 A summary of research findings

The analysis of the cloud service monitoring application over an eight-week period revealed key insights into its performance and reliability. Out of a total of 75 reported downtime incidents, the app successfully detected 64 incidents, yielding a detection rate of 85.3%. This indicates a strong but not flawless capability in identifying service disruptions. The remaining 11 incidents (14.7%) were not detected by the app, pointing to potential gaps in monitoring coverage, response timing, or infrastructure limitations. The missed downtimes were not evenly distributed; while several weeks (Weeks 3, 7, and 8) showed 100% detection, Week 5 recorded the highest number of missed incidents, suggesting variability in the app's performance. The data suggests that the app performs effectively under normal conditions but may face challenges during periods of increased activity or due to limitations in its current deployment setup, particularly since it is hosted on a local server, which may be subject to its own network or system issues. To address these findings, the report recommends improvements such as increasing monitoring frequency, enhancing logging and alerting capabilities, and considering cloud-based or hybrid hosting options for higher reliability and resilience. Overall, the app shows promise as a cloud monitoring solution, but targeted enhancements are necessary to ensure consistent, high-performance detection, especially for mission-critical services.

In the course of developing and evaluating the cloud service monitoring application, one of the notable strengths was the absence of false positives throughout the monitoring period. This reliability can be largely attributed to the implementation of synthetic transaction

monitoring, which played a key role in refining the app's alerting mechanism. Unlike traditional monitoring that relies solely on basic indicators such as ping or service port availability, the app incorporated scripted synthetic transactions that simulated actual user behaviour. These transactions involved critical workflows such as authentication requests, data retrieval, and API endpoint validations, allowing the monitoring system to test whether the application was not just responsive, but functionally operational from a user's perspective.

By validating real-world user scenarios, the app was able to distinguish between superficial system irregularities, such as momentary network lags or non-critical component failures, and genuine service disruptions. As a result, the system only triggered alerts when a simulated transaction failed, ensuring that every alert represented a meaningful incident that impacted user experience. This approach effectively filtered out noise and false alarms, which are common in simpler monitoring setups. Consequently, during the entire eight-week evaluation period, no false positives were recorded, indicating a high level of precision and dependability in the alerting mechanism.

This finding highlights the value of synthetic transaction monitoring in building a more intelligent and context-aware cloud monitoring system, especially for services where uptime alone is not a sufficient indicator of health. It reinforces the conclusion that not only did the app demonstrate a high detection rate, but it also maintained a strong balance between sensitivity and specificity, avoiding unnecessary disruptions to support or operations teams.

Chapter 5: Conclusion and Recommendations

5.1 Introduction

This chapter wraps up the study by considering the goals, highlighting the main conclusions, and providing practical suggestions. Over the course of eight weeks, the study's main objective was to assess how well a cloud service monitoring application could identify outages in real time. The study sought to ascertain whether the app satisfies reliability and performance standards by examining reported and missed downtimes in addition to key metrics like precision and F1 score.Major conclusions drawn

1. The primary objective to evaluate the app's ability to detect cloud service downtimes was successfully achieved through comprehensive data collection and statistical analysis.
2. The monitoring app showed high accuracy in detecting actual downtimes, with no false positives recorded during the observation period.
3. Simulated user transactions played a significant role in improving alert accuracy and preventing false alarms, making the system more reliable for real-world use.
4. Despite strong performance, the app missed a number of downtimes, indicating room for improvement in detection sensitivity and monitoring depth.
5. The use of precision and F1 score confirmed the app's strong performance, particularly its effectiveness in issuing correct alerts without overwhelming the user with noise.

5.2 Further Studies

While this study successfully evaluated the core performance of the cloud service monitoring app, several areas remain open for deeper exploration and enhancement. Future research could focus on the following:

1. Further studies could compare the app's performance against well-established commercial monitoring tools. This would help benchmark its capabilities and highlight areas needing improvement.

2. Research could explore how machine learning algorithms might be integrated into the app to enhance anomaly detection, pattern recognition, and adaptive thresholding based on historical data.
3. Investigating how the app performs across hybrid or multi-cloud environments could provide insights into its adaptability and performance under more complex infrastructures.
4. A focused study could examine how network delays or infrastructure limitations affect the app's ability to detect and report incidents accurately, especially when hosted on local servers.
5. Research can also investigate the user experience, including how alerts are presented and how frequently they occur, to minimize alert fatigue and improve actionable response rates.

5.3 Recommendations

1. Enhance detection logic to reduce missed downtimes
2. Promote collaboration between development, operations, and business units using insights generated by the app to improve service delivery and system design.
3. Use simulated transactions to monitor critical user flows like logins, payments, and data access to ensure end-to-end functionality, not just system uptime.
4. Even for institutions using local servers, consider cloud-based backups or failover monitoring instances to ensure resilience.
5. Use simulated transactions to monitor critical user flows like logins, payments, and data access to ensure end-to-end functionality, not just system uptime.
6. For institutions running hybrid systems, use the app to track both cloud-based and local infrastructure to maintain a full picture of operational health.
7. Train IT and operations teams on how to interpret alerts and use monitoring dashboards to make data-driven decisions.
8. Add log and metric correlation to enrich alert context and improve diagnostic insights.
9. Conduct periodic audits of missed events to improve rule accuracy.
10. Explore integration with external monitoring tools for cross-validation.

References

Alqahtani, F., Alqahtani, A. & Alharbi, K. (2021) 'Monitoring web service reliability using machine learning techniques', *International Journal of Cloud Services Research*, 18(3), pp. 1527.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2010) 'A view of cloud computing', *Communications of the ACM*, 53(4), pp. 50-58.

Avizienis, A., Laprie, J.-C., Randell, B. and Landwehr, C. (2004) 'Dependability and secure computing', *IEEE Transactions on Dependable and Secure Computing*, 1(1), pp. 1–7.

Avizienis, A., Laprie, J.C., Randell, B. & Landwehr, C. (2004) 'Basic concepts and taxonomy of dependable and secure computing', *IEEE Transactions on Dependable and Secure Computing*, 1(1), pp. 11-33.

Amazon Web Services (AWS). (2025). Amazon S3 storage service endpoint. Available at: https://s3.amazonaws.com/my-bucket-name (Accessed: 16 March 2025).

Bank for International Settlements (2021) *Cloud computing in banking: Risk management and regulatory approaches*. BIS.

Barlow, R.E. & Proschan, F. (1996). *Mathematical theory of reliability*. Philadelphia: SIAM.

Bodenstaff, L., Wombacher, A. & Reichert, M. (2011) 'Empirical validation of MoDe4SLA: Approach for managing service compositions', *14th International Conference on Business Information Systems*, no. 612, pp. 98–110. Available at: https://doi.org/10.1007/978-3-642-21863-7_9

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. & Brandic, I. (2010) 'Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility', *Future Generation Computer Systems*, 25(6), pp. 599-616.

CABS. (2024). CABS Half Year results ended 30 June 2024. Available at: https://www.cabs.co.zw/sites/default/files/CB%20011024%20%20CABS%20Half%20Year%20results%20ended%2030%20June%202024-compressed.pdf (Accessed: 16 March 2025).

Cardoso, J., Sheth, A., Miller, J., Arnold, J. & Kochut, K. (2014) 'Quality of service for workflows and web service processes', *Journal of Web Semantics*, 1(3), pp. 281-308.

Cetin, M. B., Talluri, S. & Iosup, A. (2021) 'Characterizing user and provider reported cloud failures', *arXiv preprint*, arXiv:2110.12237.

Chen, J. & Huang, Y. (2018) 'Service availability and reliability in cloud computing: A review', *Journal of Cloud Computing*, 7(1), pp. 1-15.

Chen, L., Zhang, W. & Luo, H. (2021) 'Ensuring reliability in high-traffic cloud services', *Journal of Internet Services and Applications*, 12(3), pp. 45-56.

Emeakaroha, V.C., Ferreto, T.C., Netto, M.A.S., Brandic, I. & De Rose, C.A.F. (2012) 'CASViD: Application level monitoring for SLA violation detection in clouds', *IEEE 36th Annual Computer Software and Applications Conference*, July, pp. 499–508. Available at: https://doi.org/10.1109/COMPSAC.2012.68

European Banking Authority (2022) *Guidelines on cloud outsourcing for financial institutions*. EBA.

European Central Bank (2022) *Cloud concentration risk in the banking sector*. ECB.

Financial Times (2022) *Microsoft Azure outage affects European banking services*. Retrieved from [website]

Garg, S. K., Versteeg, S. & Buyya, R. (2011) 'A framework for ranking of cloud computing services', *Future Generation Computer Systems*, 29(4), pp. 1012-1023.

Garcia, M. & Zhou, F. (2020) 'Fault tolerance in microservices architectures', *ACM Transactions on Internet Technology*, 20(4), pp. 25-40.

Gomez, L. & Torres, P. (2023) 'Cloud monitoring adoption in Latin America, North America, and Europe: A comparative analysis', *International Journal of Information Systems*, 40(3), pp. 112-129.

Gartner (2022) *State of cloud computing: Outages, risks, and trends*. Gartner.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A. & Khan, S. U. (2015) 'The rise of “big data” on cloud computing: Review and open research issues', *Information Systems*, 47, pp. 98-115.

IEEE Xplore (2020) *Evaluation metrics of service-level reliability monitoring rules*. Available at: https://ieeexplore.ieee.org (Accessed: 16 December 2024).

Johnson, R. & Green, T. (2019) 'Threshold-based monitoring in cloud services', *International Journal of Cloud Services Research*, 16(2), pp. 14-29.

Johnson, R. B. & Christensen, L. (2020). *Educational Research: Quantitative, Qualitative, and Mixed Approaches*. 7th ed. SAGE Publications, Thousand Oaks.

Kim, S., Lee, H. & Park, J. (2020) 'Distributed monitoring frameworks for cloud-based systems', *IEEE Access*, 8, pp. 105467-105478.

Köhler, A., Tavares, J. & Steinmetz, R. (2020) 'Adaptive real-time monitoring for distributed systems', *ACM SIGMETRICS Performance Evaluation Review*, 48(2), pp. 21–30.

Meng, S. & Liu, L. (2013) 'Enhanced monitoring-as-a-service for effective cloud management',
IEEE Transactions on Computers, 62(9), pp. 1705–1720. Available at: https://doi.org/10.1109/TC.2012.165

Montes, J., Sánchez, A., Memishi, B., Pérez, M.S. & Antoniu, G. (2013) 'GMonE: A complete approach to cloud monitoring', *Future Generation Computer Systems*, 29(8), pp. 2026–2040. Available at: https://doi.org/10.1016/j.future.2013.02.011

Obkio (2023) *Real-time network monitoring*. Available at: https://www.obkio.com (Accessed: 16 December 2024).

Papazoglou, M. P., Traverso, P., Dustdar, S. & Leymann, F. (2017) 'Service-oriented computing: State of the art and research challenges', *Computer*, 40(11), pp. 38-45.

Ponto, J. (2021). "Understanding and evaluating research design: A systematic approach to methodology." *Journal of Research Design and Statistics*, 14(2), pp. 150-162. https://doi.org/10.1080/21582023.2021.1892763

Reuters (2024) 'Italy's biggest bank Intesa says all tech issues resolved'. Available at: https://www.reuters.com/markets/europe/italys-intesa-says-home-banking-app-outage-linkedintense-traffic-2024-12-02/ (Accessed 21 February 2025).

Srinivas, P., Husain, F., Parayil, A., Choure, A., Bansal, C. & Rajmohan, S. (2024) 'Intelligent monitoring framework for cloud services: A data-driven approach'. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, pp. 1–11. Available at: https://www.doi.org/10.1145/nnnnnnn.nnnnnnn

Tobias, F., Fabian, K., Pascal, H., Robin, S., Dominik, S. & Andreas, K. (2021) 'Aviator: A web service for monitoring the availability of cloud services', *Nucleic Acids Research*, 49(1), pp. 1-6.

Wang, X. & Liu, Y. (2018) 'Log analysis for anomaly detection in distributed systems', *IEEE Transactions on Network and Service Management*, 15(3), pp. 756-768.

Zheng, Z., Martin, P., Brohman, M. K. & Simanta, S. (2014) 'Service monitoring for faulttolerant and reliable service-oriented architecture', *IEEE Transactions on Services Computing*, 7(1), pp. 74-85.